

Advanced Patterns in Python Programming:

From Working Code to Reviewed, Scalable Design



Syntax fluency is not enough: code must scale and survive review



Four lenses: data model, control flow, structure, performance



Reference patterns over cargo-cult copying: match pattern to problem shape



Roadmap: running example service used across all slides



How to name and defend each pattern in real code review



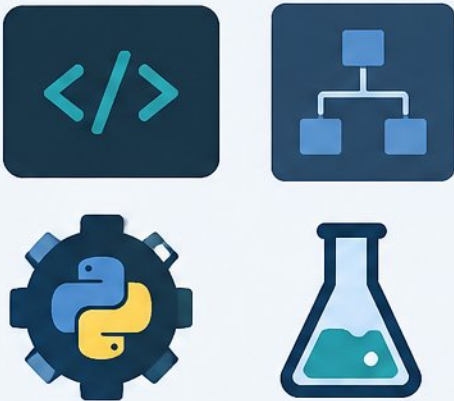
Baseline, Goals, and the Pattern Taxonomy

Presenter: [Name], [Role/Organization] · Date: October 2026

BETTER PATTERNS
BRIGHTER SOFTWARE

Baseline

Functions, classes, decorators, virtual environments, and a testing framework



Failure modes

Implicit state, leaky abstractions, accidental $O(n^2)$, untyped boundaries



Review lens

Naming, boundaries, error handling, and testability

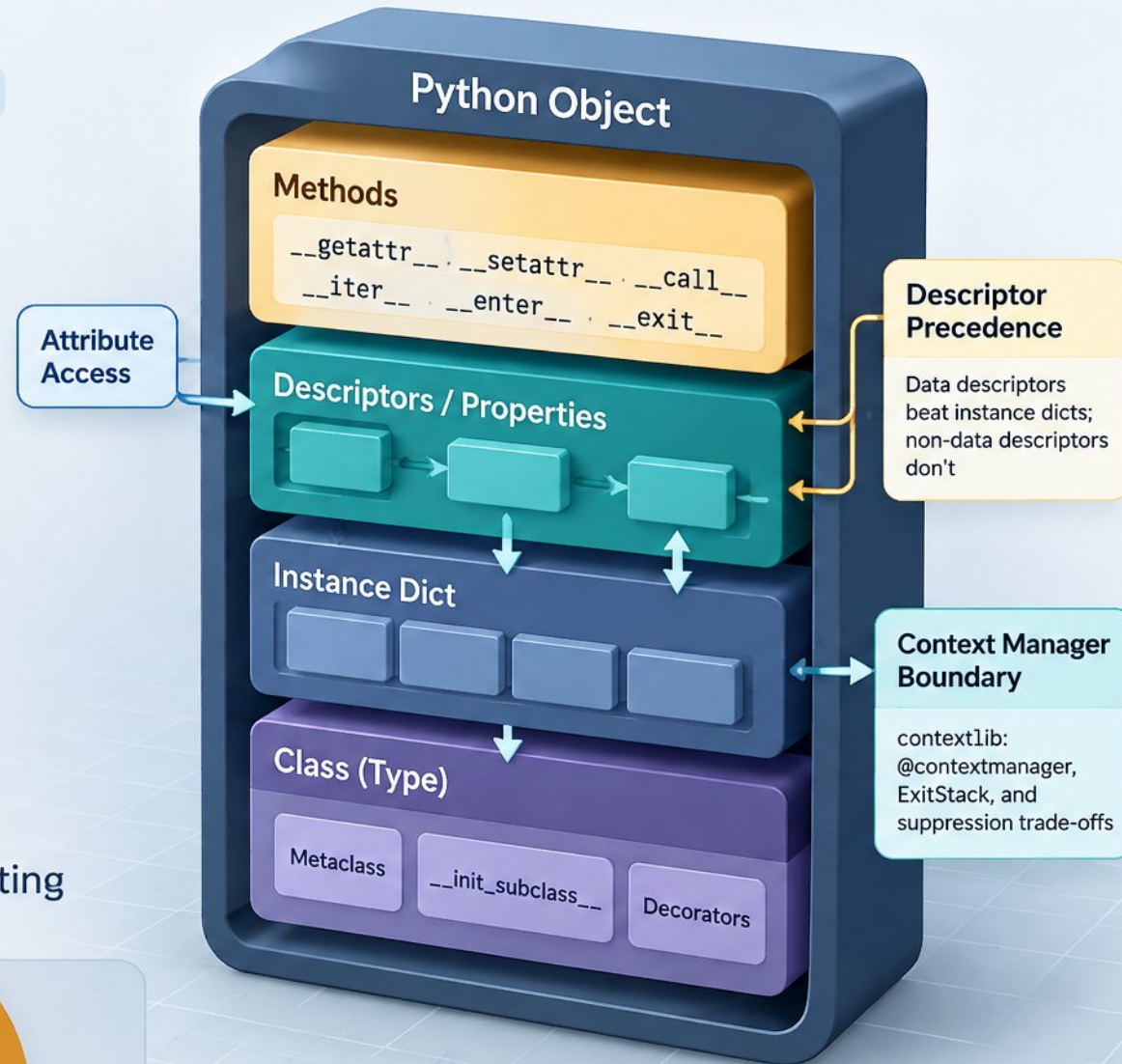


Scope

No framework tours, no language rebuilds, no pattern laundry

The Data Model as a Pattern Foundation

- Dunder protocols that matter:
`__getattr__`, `__setattr__`, `__call__`,
`__iter__`, `__enter__`, `__exit__`
- Descriptors and properties:
attribute access carrying behavior
- Precedence rule: data descriptors
beat instance dicts; non-data
descriptors don't
- Metaclasses vs `__init_subclass__`
vs decorators: pick the lightest tool
- contextlib: `@contextmanager`,
ExitStack, and suppression trade-offs
- Pitfalls: surprising attribute behavior,
MRO conflicts, singletons that fight testing



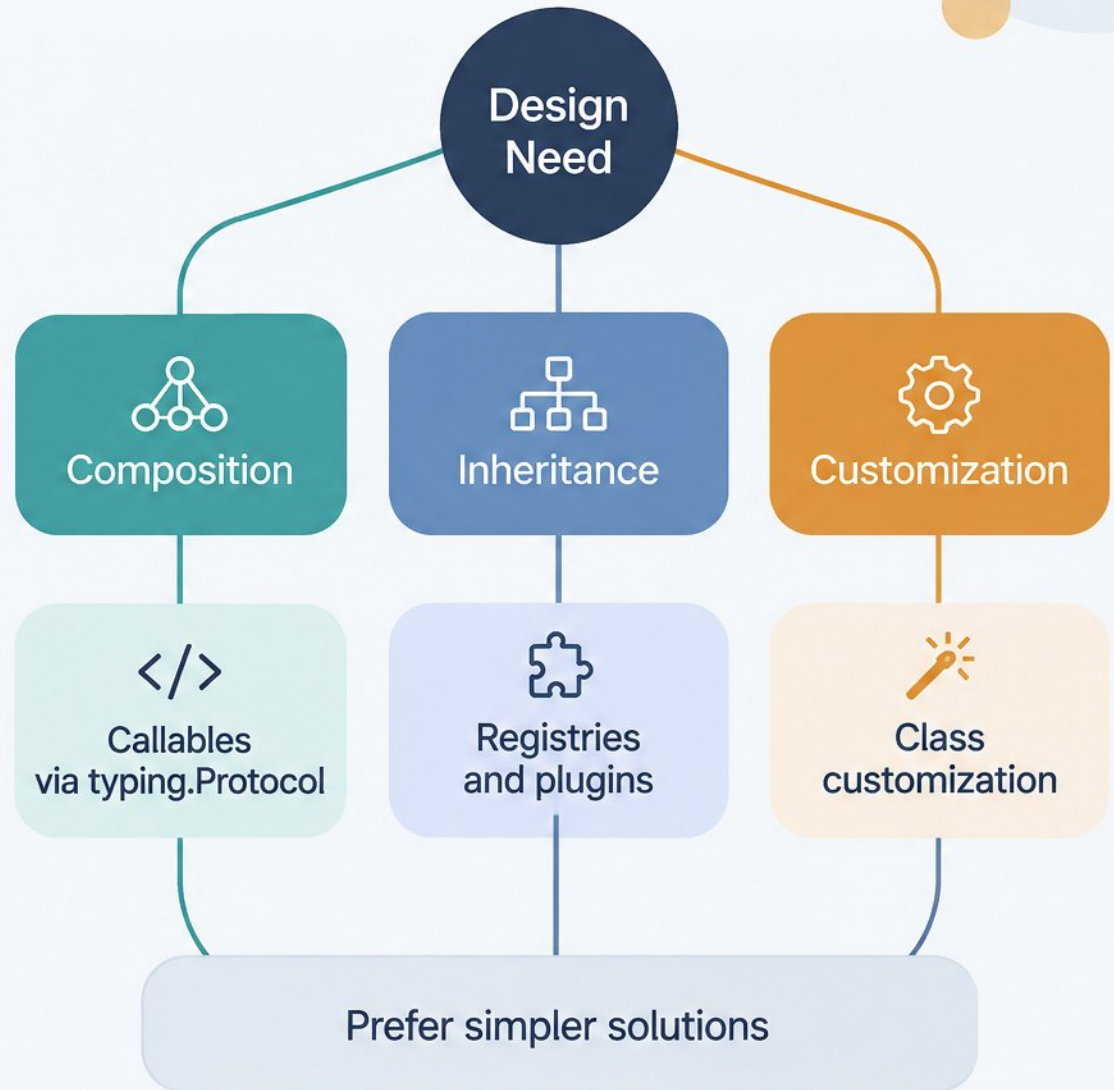
Four Lenses

Look at the data model
from different angles
to find the right pattern.



Choosing Among Composition, Inheritance, and Customization

- Mixins: keep testable, avoid MRO surprises
- Strategy and command as callables via typing.Protocol, not deep hierarchies
- Registries and plugins via entry points and importlib discovery
- Class customization: decorator, then `__init_subclass__`, then metaclass (lightest first)
- Prefer composition over decoration; a plain function often beats both



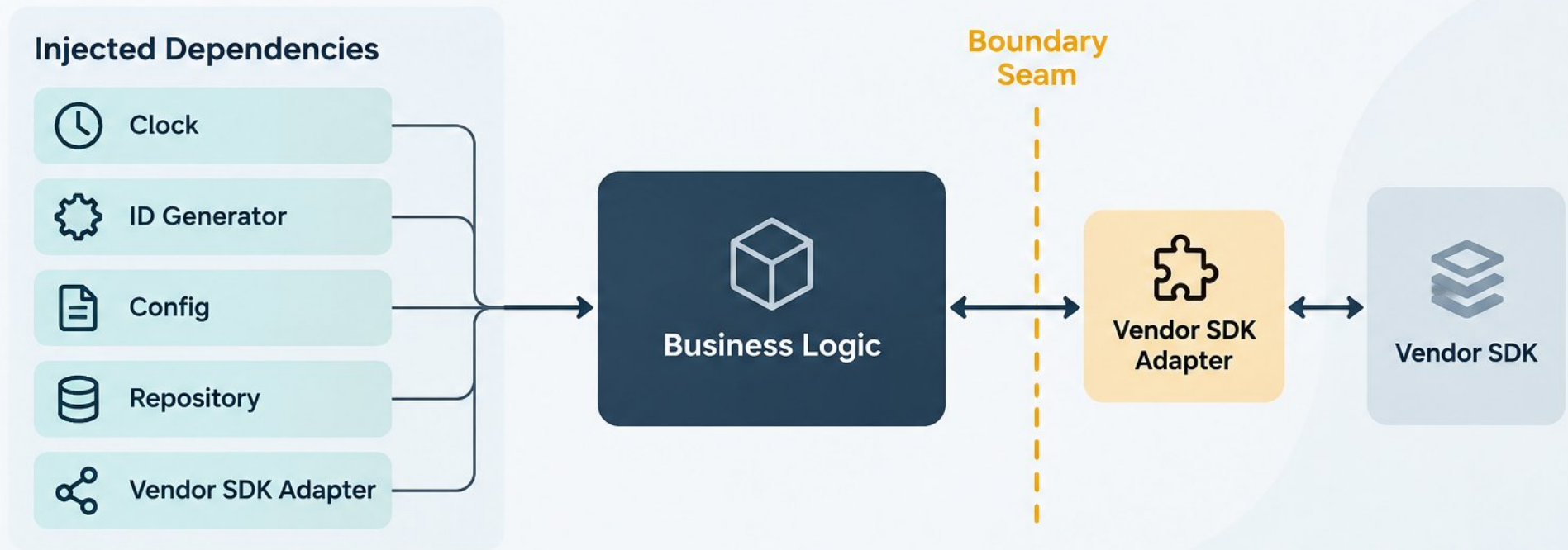
Plugin Discovery, Entry Points, and Boundaries



- Three discovery routes: naming conventions, namespace packages, and metadata entry points
- `importlib.metadata.entry_points` is the current standard discovery API
- Wrap plugin import and init in defensive error handling with clear diagnostics
- Expose only permitted objects; decide who owns plugin lifecycle
- Choose deliberately: dynamic imports, entry points, or an explicit config list

Dependency Injection Without a Framework

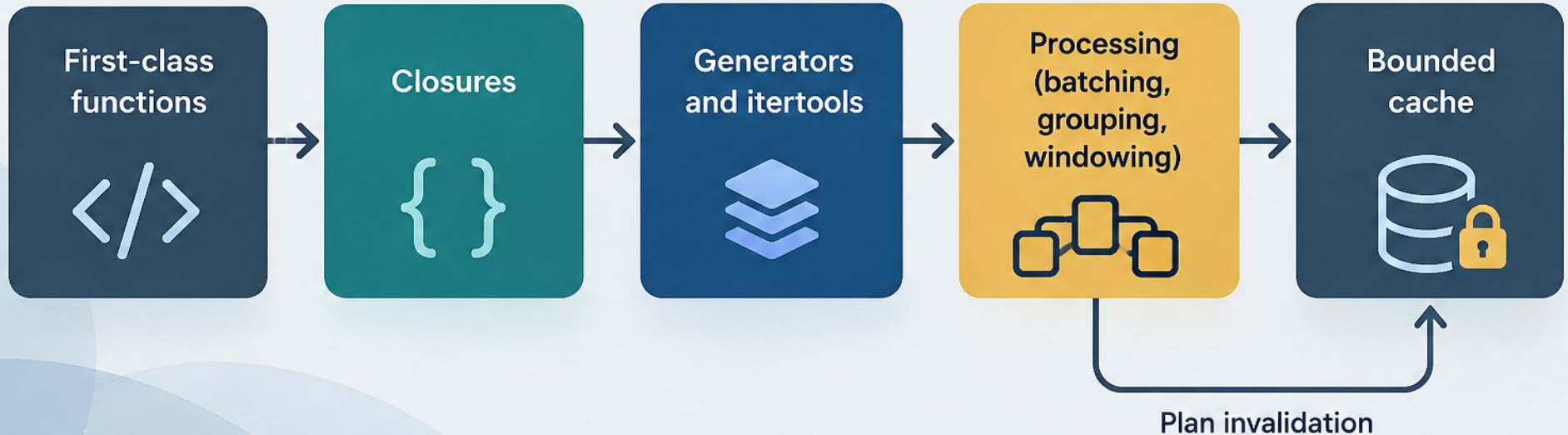
- Ports and adapters in plain Python: define explicit interfaces, own lifetimes
- Wrap vendor SDKs behind adapters so vendor shapes never leak into business code
- Stable seams for tests: inject clocks, ID generators, config, repositories, clients
- Choose constructor injection, application factories, or explicit context objects
- Weigh structural typing, ABCs, and runtime checks by cost and clarity



CLEAN BOUNDARIES
BETTER SOFTWARE

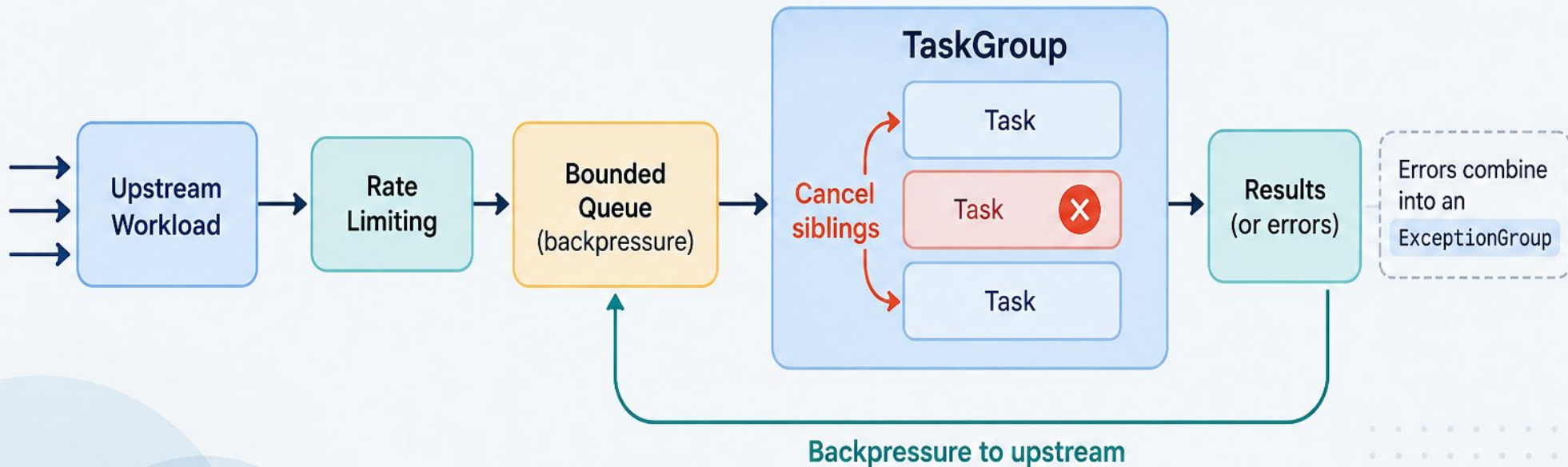
Functional Building Blocks in Production Code

- First-class functions, closures, and the late-binding trap in loops
- **functools**: partial, lru_cache, cache, singledispatch, and wraps
- Generators and itertools as lazy pipelines: batching, grouping, windowing
- Unbounded caches grow memory: bound them and plan invalidation
- Use functional style when it clarifies intent, not when it hides it



Structural Concurrency with TaskGroup and Cancellation

- Match workload shape to asyncio, threads, processes, or task queues
- TaskGroup: tasks added via `create_task()`, all awaited on context exit
- First failure cancels siblings; errors combine into an `ExceptionGroup`
- Avoid swallowing `CancelledError`; it breaks timeout and group cancellation
- Bound queues for backpressure; keep blocking calls off the event loop



Async Refactoring, **Observability**, and Async Anti-Patterns

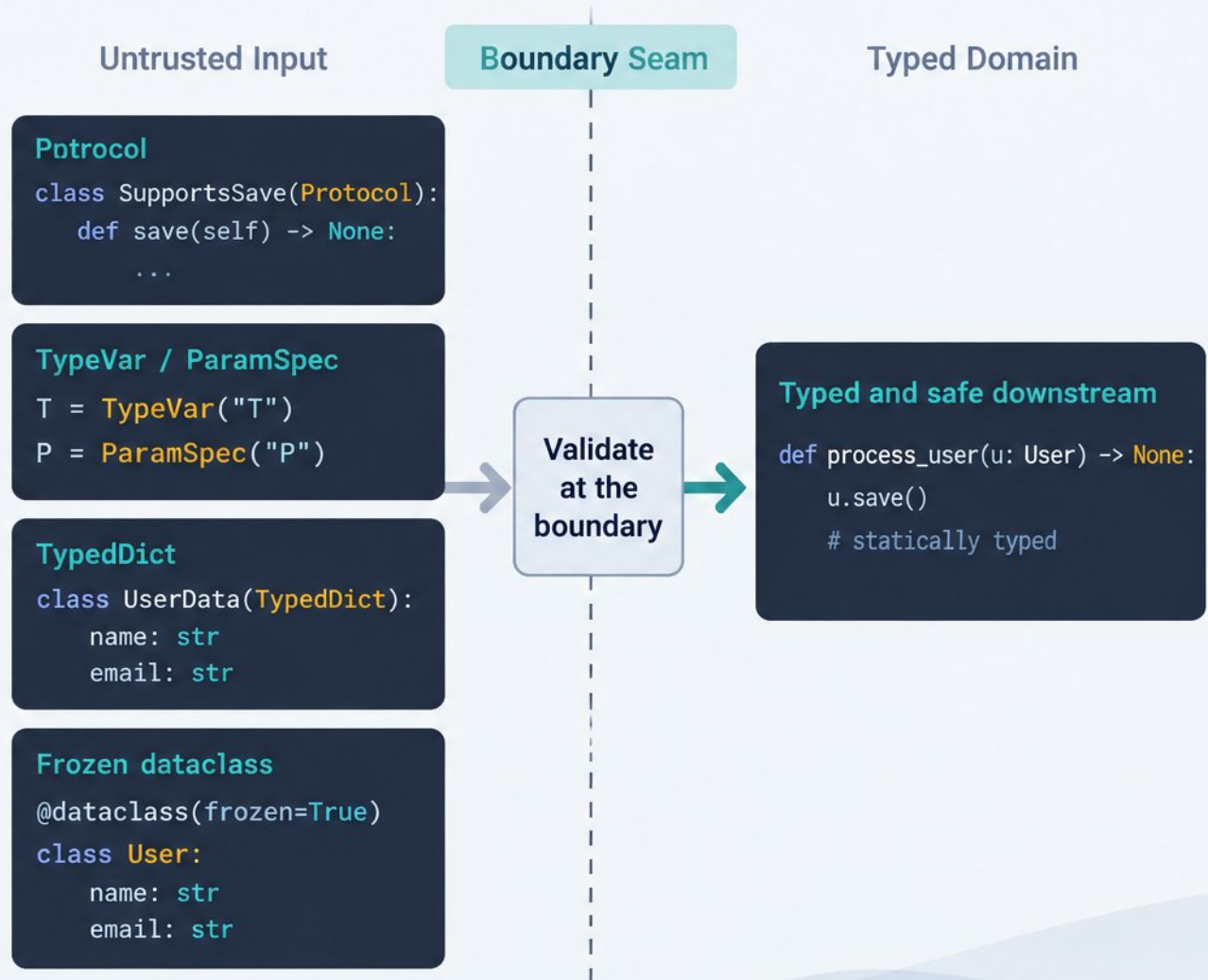
- Blocking libraries inside async def stall the entire event loop
- Untracked background tasks can vanish: weak refs, lost exceptions
- Swallowed CanceledError breaks TaskGroup and asyncio.timeout()
- Shared mutable state across tasks needs locks or coordination
- Set correlation IDs, name tasks, log in structured form
- Propagate timeouts and cancellation; sometimes skip async entirely



Typing and Data Modeling at the Boundaries

VALIDATE AT THE EDGE. STRONGER CODE INSIDE.

- 1 Protocols for structural subtyping: annotate arguments, not concrete classes
- 2 TypeVar for reusable generics; ParamSpec for decorators preserving signatures
- 3 TypedDict and frozen dataclasses model untrusted input at the seam
- 4 Validate once at the boundary; let static types govern everything downstream
- 5 Gradual typing: annotate seams first, keep Any contained and visible

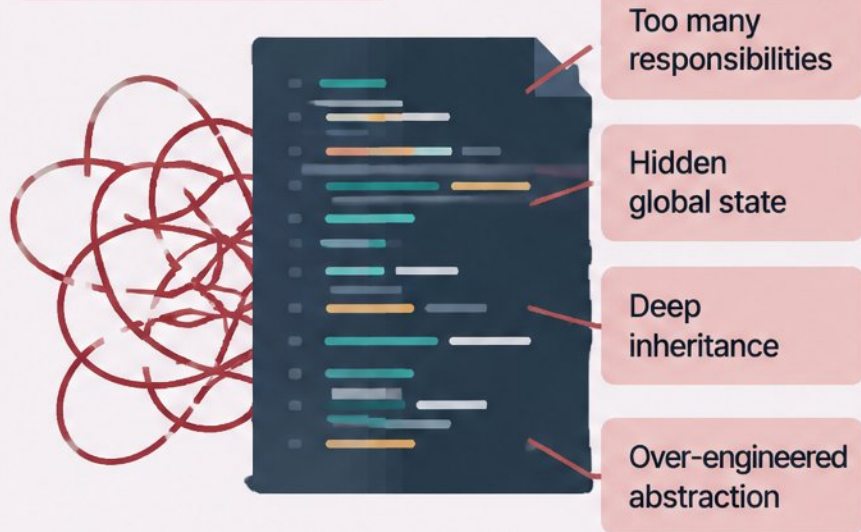


Anti Patterns and Refactoring to Stronger Designs

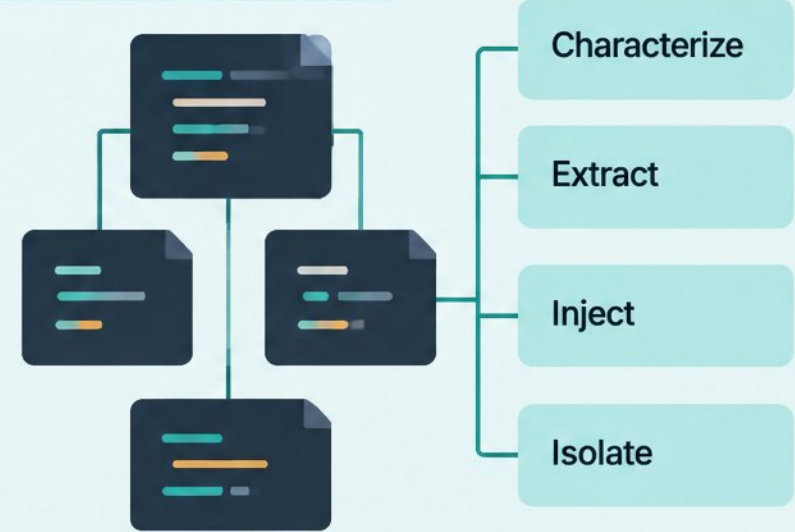
From fragile code to maintainable, testable, and adaptable designs



Anti-Patterns



Stronger Design



Key Points

- ! God objects and huge modules: too many responsibilities, frequent merge conflicts
- ! Hidden global state leaks dependencies and breaks test isolation
- ! Deep inheritance and MRO surprises cause spooky action at a distance

Moving Forward

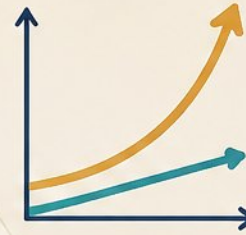
- Over-engineered abstraction adds friction where a plain function suffices
- Refactoring moves: characterize, extract, inject, isolate, pin behavior with tests
- Review heuristics: flag repeated edits, brittle tests, and vendor SDK leakage

Performance, Profiling, and Optimization Patterns

WRITE
MEASURE
UNDERSTAND
IMPROVE



Measure first:
cProfile, timeit, and
sampling profilers,
never guess



Complexity in real code:
accidental quadratic
loops and repeated
work

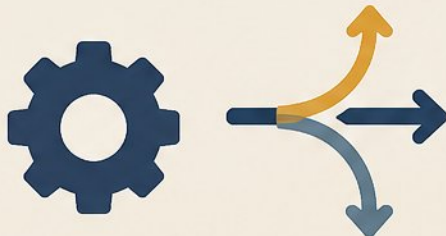


Caching layers:
memoization,
request-scoped
caches, and
invalidation



Memory patterns:
__slots__, generators,
streaming over
bulk loading

FOUR
LENSES



Acceleration decisions:
C extensions, Cython/Rust, and when not to

Pattern Language for Reviews, Teaching, and Team Standards



Name patterns and trade-offs explicitly in review comments



Checklist: boundaries, errors, concurrency, typing, testability



Teach with examples, counterexamples, and deliberate practice



Codify conventions lightly; keep room for innovation



Adopt via pilots, docs, and honest metrics—no vanity numbers

Code Review Checklist

- ☐ Boundaries
- ☐ Errors
- ☐ Concurrency
- ☐ Typing
- ☐ Testability

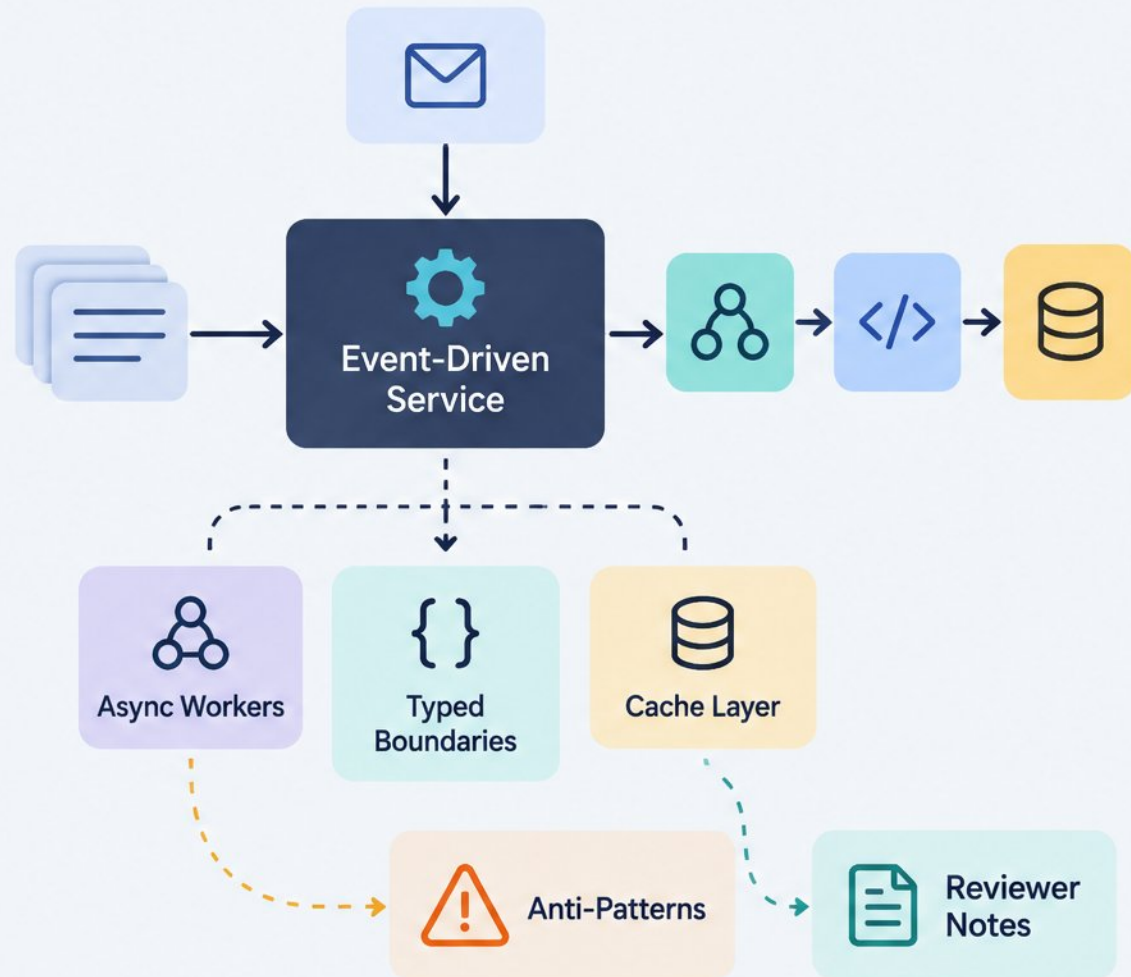
Patterns & Trade-offs

-
-
-



Capstone Workshop: Applied Patterns in a Small Service

- Build a small event-driven service with a data pipeline
- Apply composition, TaskGroup concurrency, typing, and caching
- Hunt anti-patterns you introduced during the exercise
- Review as a reviewer; justify each pattern choice
- Debrief: what to carry into real repos next week



PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/826b7aa1/public